```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 18 11:40:29 2016

@author: glennthompson
"""

import obspy.signal.trigger as trigger
import random
import matplotlib.pyplot as plt

def tune_sta_lta(tr, algorithm, tsignalstart, tsignalend, ntrys):
    '''tune_sta_lta Tune (optimize) STA and LTA window lengths for
    STA/LTA algorithms to maximise the characteristic STA:LTA func

    Inputs:
        tr - a trace object
        algorithm - the STA/LTA method to use from
               ['classic_sta_lta', 'z_detect', 'recursive_sta_lta'
        tsignalstart, tsignalend - the number of seconds into the
        thresh_on, thresh_off - used in plot_trigger plot only
        ntrys = number of STA/LTA window combinations to try

    Outputs:
        a list (called result) containing:
            sta_best - length of best STA window (seconds)
            lta_best - length of best LTA window (seconds)
            staltaratio_best - the characteristic function returne

    To do:
        Support filtering?

Example:

# import the tune_sta_lta package
import sys
sys.path.append('/path/to/directory/containing/tune_sta_lta.py')
import tune_sta_lta as tsl

# read a seismogram into a trace object
from obspy.core import read
st = read("https://examples.obspy.org/ev0_6.a01.gse2")
st = st.select(component="Z")
tr = st[0]

# call the tune_sta_lta function
```

```
algorithm = 'classic_sta_lta'
tsignalstart = 30.0
tsignalend = 40.0
ntrys = 100

tsl.tune_sta_lta(tr, algorithm, tsignalstart, tsignalend, ntrys)
    '''
    plt.close('all')
    df = tr.stats.sampling_rate
    # LTA window cannot be longer than 30% of the overall time win
    MAX_SECONDS = int(0.3 * tr.stats.npts / df)

    print '\nAlgorithm: %s' % algorithm
    m = 1.0
    sta_best = 0
    lta_best = 0
    for count in range(1,ntrys):
        # draw STA window length from uniform distribution of 0.1
        sta_seconds = round(random.random()*10, 1)

        # draw LTA window length uniform distribution of 2 to 10 t
        lta_seconds = sta_seconds * round(2+random.random()*8,0)
        if lta_seconds > MAX_SECONDS:
           lta_seconds = MAX_SECONDS

        staltaratio = float("-inf")
        if algorithm == 'classic_sta_lta':
            staltaratio = trigger.classic_sta_lta(tr.data, int(sta
        elif algorithm == 'z_detect':
            staltaratio = trigger.z_detect(tr.data, int(lta_second
        elif algorithm == 'recursive_sta_lta':
            staltaratio = trigger.recursive_sta_lta(tr.data, int(s
        elif algorithm == 'carl_sta_trig':
            staltaratio = trigger.carl_sta_trig(tr.data, int(sta_s
        elif algorithm == 'delayed_sta_lta':
            staltaratio = trigger.delayed_sta_lta(tr.data, int(sta
        maxstalta = max(staltaratio[int(df*tsignalstart):int(df*ts
        # If the max STA/LTA ratio is better than what we already
        # update and print these new settings
        if maxstalta > m:
            m = maxstalta
            sta_best = sta_seconds
            lta_best = lta_seconds
            staltaratio_best = staltaratio
            print ("sta_seconds=%.1f lta_seconds=%.1f max(staltara
```

```
   # now we have captured the best STA/LTA settings
   # run them again & plot triggers
   #trigger.plot_trigger(tr, staltaratio_best, thresh_on, thresh_o

#=================================================================
#         fig,axarr = plt.subplots(2,1,sharex=True)
#         fig.suptitle(algorithm)
#         axarr[0].plot(tr.data)
#         axarr[0].set_title('Time series' )
#         axarr[1].plot(staltaratio_best)
#         axarr[1].set_title('STA/LTA ratio, STA=%.1fs, LTA=%1.fs'
#         plt.show(block=False)
#=================================================================
    result = [sta_best, lta_best, staltaratio_best]
    return result
```